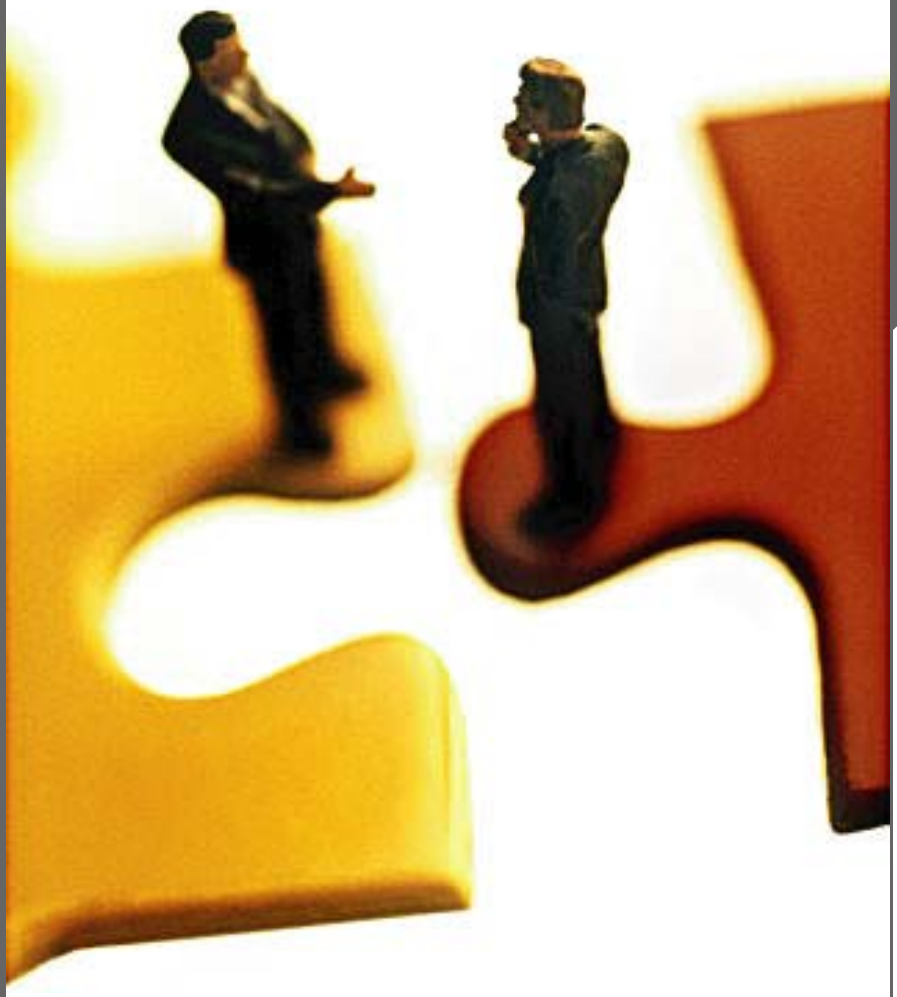


Interakt

PROFESSIONAL WEB TOOLS

KTML JavaScript

API Documentation

Table of Contents

Overview....3
Prerequisites....4
Requirements....4
Typographic Conventions....4
Intended Audience....5
Quick Install Guide....6
KTML Architecture....7
KTML Instantiation....8
Including the JavaScript Required Files and the KTML Initialization Global Call....8
KTML Object Instantiation....10
Setting the KTML Object Properties....11
KTML Application Programming Interface....17
The KTML Object Methods....17
KTML Toolbar....19
The Selection Mechanism....19
Built-in Toolbar Buttons....19
Creating a Custom Toolbar Button....20
Adding a Custom Toolbar Button....22
KTML Plug-in Mechanism....23
Plug-in Architecture....23
Creating a Plug-in Sample....23
Conclusions....28
Further Reading....28

Overview

This document aims to teach you how to install and customize the **KTML** online visual HTML editor. You will learn the **KTML** architecture and how to include and customize such an advanced control into your web pages. The application is not server-side, but is very extensible as you can add your own plug-ins depending on the web server you decide to use.

KTMLis an online visual HTML editor designed for publishers who need to edit web content fast. It allows easy content formatting using its advanced table manipulation features based on the live property inspector. Unlike other online editors on the market, our product supports most platforms and browsers, providing the user with advanced text and paragraph formatting capabilities, as well as CSS support, and easy HTML tag selection.

Prerequisites

Requirements

You can edit web content with **KTML** from anywhere, without having to download or install any special program (supposing you have already installed **KTML** on your content management system). All you need is a computer with an operating system and an web browser. This can be your local computer, or any other computer.

The **KTML** requirements are:

- ***Web browsers:***
  - **Microsoft Internet Explorer 6.0** or higher
  - **Mozilla 1.4** or higher
- ***Operating systems:***
  - **Windows 98, ME, NT, 2000, XP**
  - **Mac OSX**

Typographic Conventions

The notations and text formats used in this document are found below:

- Site page: mono-spaced italic "*index.htm*"
- Application button or menu: bold font "**Button**"
- User interface elements: bold italic font "***Property Inspector***"
- HTML source tags and attributes: italic font "<*script*>"
- Source code : mono-spaced font "`<script src="init.js"></script>`"

Intended Audience

This document addresses to web programmers who want to include and extend the **KTML** visual editor features and to create their own plug-ins in a certain language depending on the chosen web server.

By including the advanced **KTML** control, the programmers will offer their clients the possibility to edit content visually.

Quick Install Guide

To install **KTML** follow the steps briefly presented below:

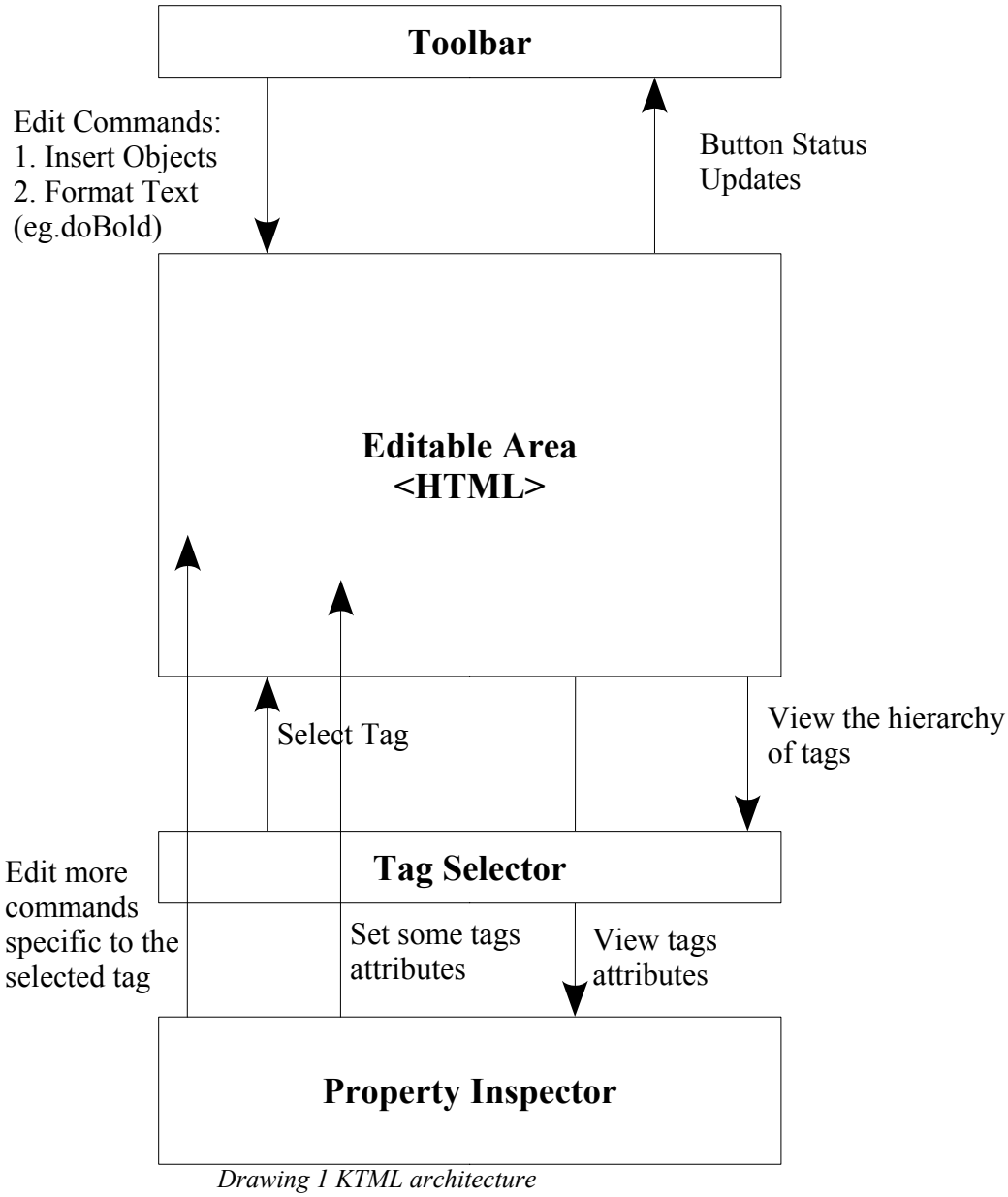
1. Unzip the **KTML** package and install the **ktml** folder in your site root.
2. Create a new file in your site root (named *index.htm*) where you should:
    1. Include the JavaScript required files and the KTML initialization global call.
    2. Include a KTML object instantiation.
    3. Set the KTML object properties.
    4. Include the built-in toolbar buttons.
    5. Add your own plug-ins.

KTML Architecture

KTML was designed to satisfy the needs for a non-server-side HTML visual editor. Our product was developed in JavaScript but is very extensible allowing you to extend the features and to add your own plug-ins depending on the chosen web server.

The KTML object has 4 areas communicating amongst themselves:

1. ***Toolbar*** – containing the buttons allowing to apply commands to the edited content.
2. ***Editable Area*** – allowing to insert HTML code visually.
3. ***Tag Selector*** – used to display and select various tags.
4. ***Properties Inspector*** – used to display and set the properties for the selected item.



Drawing 1 KTML architecture

The editing commands are executed by using API methods belonging to the KTML object such as: ***doBold***, ***doUndo***, ***openColorPicker***, ***getSelectedNode***, ***insertObject***, ***insertText***, ***applyCSSStyle*** and others.

Some commands have dialog windows that open when the corresponding toolbar button is clicked on and that allow setting some additional properties for the inserted object.

KTML Instantiation

In this chapter you will learn how to include one or more KTML objects into your web page. Say you name your page `index.htm` and you place it in your site root.

Including the JavaScript Required Files and the KTML Initialization Global Call

Open the `index.htm` page and include the following code lines inside the `<head>` tag:

```
<script src="ktml/includes/ktedit/init.js"></script>
<script id="ktml_global_jsvariables">
      initKTML("ktml/", "php", "francais");
      if(is.valid && is.mozilla){
            includeJS(DIR_DEPTH+"includes/ktedit/mozilla_ie_compat.js");
      }
</script>
<script src="ktml/includes/ktedit/utils3.js"></script>
<script src="ktml/includes/ktedit/ktml.js"></script>
```

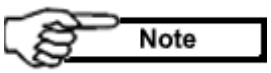
As you can see, you should insert 4 <*script*> tags:

1. The `<script src="ktml/includes/ktedit/init.js"></script>` code line will include the `init.js` file. In this file a browser check is performed in order to verify if it is valid (**Internet Explorer 6.0** or higher or **Mozilla 1.4** or higher). The `init.js` file also includes a style sheet for the KTML object.

2. The next <*script*> tag will perform the KTML initialization global call.
   - The `initKTML` function has the following syntax:

     ```
     function initKTML(p_dirDepth, p_ext, p_globalUILanguage)
     ```

     where:
     - ***p_dirDepth*** – is the relative path to the folder where KTML is installed from the `index.htm` file.
     - ***p_ext*** – is the file extension for the plug-ins corresponding to the chosen web server.
     - ***p_globalUILanguage*** – is the KTML display language. If the display language is also set in the local call (when the KTML object is instantiated) by calling the `setUILanguage` function, the language set in the global call is ignored.
   - This <*script*> tag will also set some other global variables.



As said above, ***p_dirDepth*** is the relative path from the `index.htm` file to the **ktml** folder with a trailing slash.

Example: if the file `index.htm` is located in a folder named **deep**, and the **ktml** is locted in the root folder

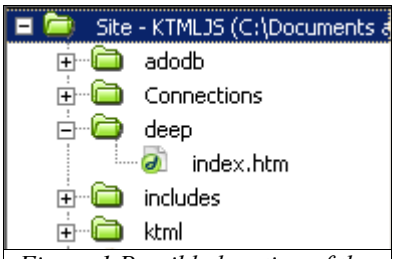


Figure 1 Possible location of the index.htm file

then the path would be:

```
<script src="../ktml/includes/ktedit/init.js"></script>
<script id="ktml_global_jsvariables">
            initKTML("../ktml/", "php", "francais");
```

3. The third <*script*> tag will include the `utils3.`js file which contains some helper functions.

4. The last <*script*> tag will include the `ktml.`js file which contains the KTML class – this is where the KTML object is defined.

```
<html>
<head>
<title>KTML JSOnly Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

<script src="ktml/includes/ktedit/init.js"></script>
<script id="ktml_global_jsvariables">
  initKTML("ktml/", "php", "francais");
  if(is.valid && is.mozilla){
    includeJS(DIR_DEPTH+"includes/ktedit/mozilla_ie_compat.js");
  }
</script>
<script src="ktml/includes/ktedit/utils3.js"></script>
<script src="ktml/includes/ktedit/ktml.js"></script>

</head>
```

Figure 2 Including the JavaScript Required Files and the KTML Initialization Global Call

KTML Object Instantiation

The KTML object should be attached to a <*textarea*> from which it will inherit the dimensions.

Therefore, inside the <*body*> tag, insert a form and a text-area:

```
<form>
<textarea name="cont_cnt" id="cont_cnt" cols="50" rows="5"
style="width:700px; height:400px">Enter<b>html</b>text</textarea>
```

As you can see, we named the text-area ***cont_cnt***.

The next step is to create a KTML object instance:

```
<script id="ktml_cont_cnt_instance">
                ktml_cont_cnt = new Ktml('cont_cnt');
```

There is a constraint regarding the name of the KTML object instance: its name should always begin with ***ktml_***.

In our case, we named it ***ktml_cont_cnt*** in order to mention also the corresponding text-area name. This naming rule is very useful for the case where you want to include more KTML instances on the same page attached to different text-areas.

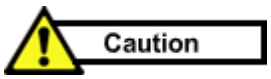
```
<script src="ktml/includes/ktedit/init.js"></script>
<script id="ktml_global_jsvariables">
  initKTML("ktml/", "php", "francais");
  if(is.valid && is.mozilla){
    includeJS(DIR_DEPTH+"includes/ktedit/mozilla_ie_compat.js");
  }
</script>
<script src="ktml/includes/ktedit/utils3.js"></script>
<script src="ktml/includes/ktedit/ktml.js"></script>

</head>

<body>
<form>
<textarea name="cont_cnt" id="cont_cnt" cols="50" rows="5" style="width:700px; height:400px">Enter<b>html
</b>text</textarea>
<script id="ktml_cont_cnt_instance">
  ktml_cont_cnt = new Ktml('cont_cnt');
```

Figure 3 Creating a KTML object instance

Next, inside this <*script*> tag you should call the functions that will set the KTML object properties.



Pay attention not to forget to call the `paint()` function after the calls of the functions that set the properties and before those that create custom toolbar buttons.

Setting the KTML Object Properties

For each KTML object instance on the page you can set the properties by calling some functions that will be explained further on. These functions should be called inside the <*script*> tag where the KTML object is instantiated.

The setUseIntrospection Function

This function activates or deactivates the ***Property Inspector***. By default, it is set to true – meaning that if it is not called, the ***Property Inspector*** will be displayed.

Function Syntax

```
function Ktml_setUseIntrospection(useIntrospection);
```

- ***useIntrospection*** – boolean – if `false`, the ***Property Inspector*** will not be displayed. By default it is set to `true`.

Function Example

```
ktml_cont_cnt.setUseIntrospection(false);
```

The setDebug Function

This function sets the DEBUG mode for the KTML object. The debug messages will be displayed in a new window. The default value is `false`, so if this function is not included when setting the KTML object properties, the DEBUG window will not be displayed.

Function Syntax

```
function Ktml_setDebug(pDebug)
```

- ***pDebug*** – boolean – if `true`, the DEBUG window will be displayed. By default it is set to `false`.

Function Example

```
ktml_cont_cnt.setDebug(true);
```

In the below image we present the DEBUG window when no error occurs:

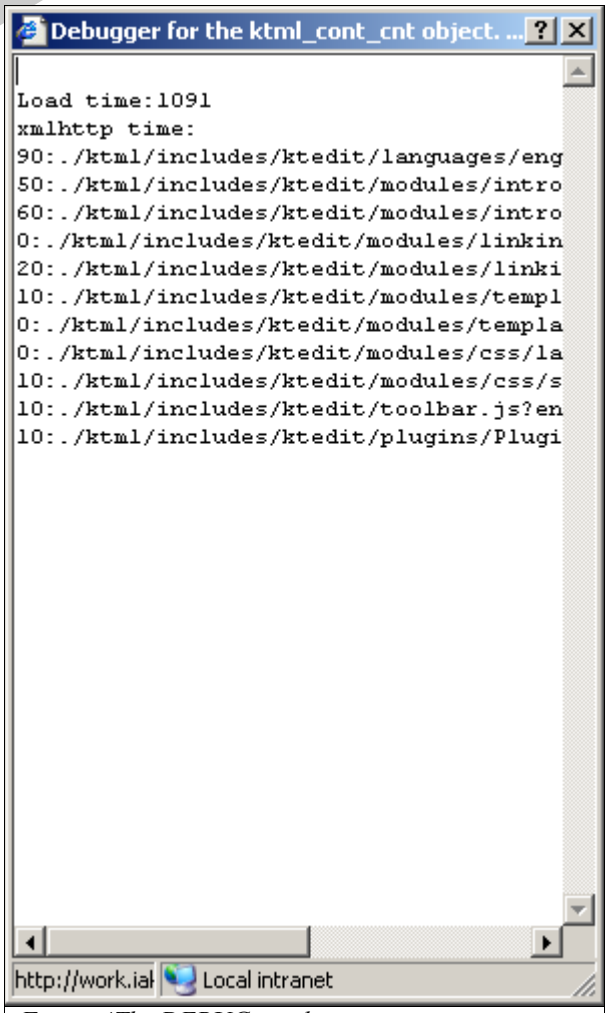


Figure 4The DEBUG window

The setSize Function

This function allows you to set the KTML editable area size. Normally the KTML object gets its size from the text-area it is attached to if no `setSize` function is called. When the size is set both in the text-area definition and by the `setSize` function, the `setSize` function has priority. It should also be mentioned that the actual width will be forced by the toolbar width, which may be larger than the specified KTML width.

Function Syntax

```
function Ktml_setSize(width, height)
```

- ***width*** – the KTML editable area's width in pixels.
- ***height*** – the KTML editable area's height in pixels.

Function Example

```
ktml_cont_cnt.setSize("600px","300px");
```

The setReadOnly Function

This function allows you to set whether the KTML editable area should be editable or read-only. By default the area will be editable and you will not even have to call the function inside the <*script*> tag where the KTML object is instantiated.

Function Syntax

```
function Ktml_setReadOnly(readOnly)
```

- ***readOnly*** – boolean – if `true`, the KTML editable area will be read-only. By default it is set to `false`.

Function Example

```
ktml_cont_cnt.setReadOnly(true);
```

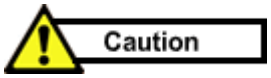
The setToolbarButtons Function

This function allows you to choose the built-in buttons that will be displayed in the KTML toolbar. The buttons can be arranged in more than one row within the toolbar. For that you should also include `"BREAK"` in the list of buttons. It will insert a break after the button where it is inserted.

Function Syntax

```
function Ktml_setToolbarButtons(buttonsVariable)
```

- ***buttonsVariable*** – it is a vector formed by the name of the built-in buttons that will be displayed in the the KTML toolbar.



These are only the built-in buttons, the custom toolbar buttons will be defined after the `paint()` function call.

For the list of all built-in buttons search the `ktmlButtons` variable in the `toolbar.js` file. For more information, please refer to the **KTML Toolbar** chapter further in this document.

Function Example

```
ktml_cont_cnt.setToolbarButtons(new Array
("Introspection","Cut","Copy","Paste","Undo","Redo","InsertImage","InsertTab
le","BulletList","NumberedList","Indent","Outdent","HorizontalLine","AlignLe
ft","AlignCenter","AlignRight","AlignJustify","ToggleVisible","ToggleEditMod
e","CleanHTMLContent","About","Help","BREAK","HeadingList","Style
List","Font
Type","FontSize","InsertLink","Bold","Italic","Underline","ForegroundColor",
"BackgroundColor"));
```

The setUILanguage Function

This function sets the KTML display language. The default user interface language is set globally by the `initKTML` function. But the `setUILanguage` function overwrites the global language settings. This function is very useful when wanting to include more KTML instances on the same page.

Function Syntax

```
function Ktml_setUILanguage(uiLang)
```

- ***uiLang*** – can have one of the following values: ***english***, ***francais***, ***dutch***, ***espagnol***, ***german***, ***norwegian***.

At the moment, **KTML** supports 6 languages. The language resources files can be found in the following folders:

- */ktml/includes/ktedit/languages/*
- */ktml/includes/ktedit/modules/introspection/languages/*
- */ktml/includes/ktedit/tags/a/languages/*
- */ktml/includes/ktedit/tags/img/languages/*
- */ktml/includes/ktedit/tags/none/languages/*
- */ktml/includes/ktedit/tags/table/help/*
- */ktml/includes/ktedit/tags/table/languages/*
- */ktml/includes/ktedit/tags/td/help/*
- */ktml/includes/ktedit/tags/td/languages/*
- */ktml/includes/ktedit/tags/tr/help/*
- */ktml/includes/ktedit/tags/tr/languages/*
- */ktml/includes/ktedit/help/*

Function Example

```
ktml_cont_cnt.setUILanguage("francais");
```

The setAutoFocus Function

This function allows you to automatically insert the cursor inside the KTML editable area even if on the page there are multiple controls.

Function Syntax

```
function Ktml_setAutoFocus(autoF)
```

- ***autoF*** – boolean – if `false`, the auto-focus will not be set. By default it is set to `true`.

Function Example

```
ktml_cont_cnt.setAutoFocus(true);
```

The setXHTML Function

This function instructs the KTML object to save the edited content as valid XML to the database. In ***View Source***

mode, the edited content will not be displayed as XML because it would take a lot of time and resources, but it will be saved as XML in the database.

Function Syntax

```
function Ktml_setXHTML(saveXhtml)
```

- ***saveXhtml*** – boolean – if `false`, the content will not be saved as XML in the database, but HTML. By default it is set to `false`.

Function Example

```
ktml_cont_cnt.setXHTML(true);
```

The setModuleProperty Function

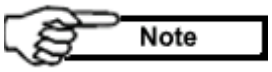
This function sets the properties for possible KTML modules. For the moment, in **KTML** only the CSS module is available, therefore this function sets only this module's properties. This function will be called later on by the corresponding function `getModuleProperty`.

Function Syntax

```
function Ktml_setModuleProperty(moduleName, propertyName, propertyValue)
```

Function Example

```
ktml_cont_cnt.setModuleProperty("css", "pathToStyle",
"ktml/includes/styles/KT_styles.css");
```



The path to the style sheet is relative to the CSS file (from the current file).

The paint() Function

This function will output all HTML required by the KTML object. It should be called after all KTML properties have been set and before the custom toolbar buttons have been defined.

Function Syntax

```
function Ktml_paint()
```

Function Example

```
ktml_cont_cnt.paint();
```

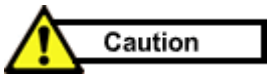
```
      <textarea name="cont_cnt" id="cont_cnt" cols="50" rows="5"><?php echo $Recordset1->Fields('cont_cnt'
); ?></textarea>
<script id="ktml_cont_cnt_instance">
  ktml_cont_cnt = new Ktml('cont_cnt');
  ktml_cont_cnt.setDebug(false);
  ktml_cont_cnt.setSize("500px", "200px");
  ktml_cont_cnt.setReadOnly(false);
  ktml_cont_cnt.setToolbarButtons(new Array("Cut","Copy","Paste","Undo","Redo","InsertImage","InsertTable"
,"BulletList","NumberedList","Indent","Outdent","HorizontalLine","AlignLeft","AlignCenter","AlignRight",
"AlignJustify","ToggleVisible","ToggleEditMode","CleanHTMLContent","About","Help","BREAK","HeadingList",
"Style List","Font Type","FontSize","InsertLink","Bold","Italic","Underline","ForegroundColor",
"BackgroundColor"));
  ktml_cont_cnt.setUseIntrospection(false);
  ktml_cont_cnt.setAutoFocus(true);
  ktml_cont_cnt.setXHTML(true);
  ktml_cont_cnt.setModuleProperty("css", "pathToStyle", "ktml/includes/styles/KT_styles.css");

  ktml_cont_cnt.paint();

ktml_cont_cnt.addToolbarButton( new ToolbarButton("st", "st", "st", null, null, "img", null,
KT_JS_FUNCTION, customFunction));

function customFunction(){
  var selectedNode  = ktml_cont_cnt.getSelectedNode();
  selectedNode.style.textDecoration="line-through";
}
```

Figure 5 Calling the paint() function



After setting all the KTML object properties, it is **mandatory** to call the `paint()` function, otherwise the KTML editor will not be displayed.

KTML Application Programming Interface

The **KTML API** (Application Programming Interface) represents a set of methods belonging to the KTML object. These methods will be used in order to take actions on the edited content.

The KTML Object Methods

The KTML object methods are defined in the `ktml.js` file.

```
Ktml.prototype.doBold = Ktml_doBold;
Ktml.prototype.doItalic = Ktml_doItalic;
Ktml.prototype.doUnderline = Ktml_doUnderline;

Ktml.prototype.doUndo = Ktml_doUndo;
Ktml.prototype.doRedo = Ktml_doRedo;
Ktml.prototype.openColorPicker = Ktml_openColorPicker;

Ktml.prototype.getSelectedNode = Ktml_logic_getSelectedNode;
Ktml.prototype.removeSelectedNode = Ktml_logic_removeTag;

Ktml.prototype.insertObject = Ktml_insertObject;
Ktml.prototype.insertText = Ktml_insertText;
Ktml.prototype.insertHTML = Ktml_insertHTML;

Ktml.prototype.getHTMLContent = Ktml_getHTMLContent;
Ktml.prototype.getXHTMLContent = Ktml_getXHTMLContent;

Ktml.prototype.applyCSSStyle = Ktml_logic_InsertStyle;
Ktml.prototype.applyHeadingStyle = Ktml_logic_InsertHeading;
Ktml.prototype.openInsertTableDLG = Ktml_openInsertTableDLG;

Ktml.prototype.getAllSelections = Ktml_getAllSelections;

Ktml.prototype.addUndoContext = Ktml_addUndoContext;

Ktml.prototype.expandSelectionToParentNode = Ktml_expandSelectionToParentNode;
```

Figure 6 The definition of the KTML object methods

You will find below a list of all defined KTML object methods:

Method name	Action to perform
doBold	Applies a bold format to the selected text.
doItalic	Applies an italic format to the selected text.
doUnderline	Applies an underlined format to the selected text.
doUndo	Reverts the edited document to previous state.
doRedo	Re-does the last undo action.
addUndoContext	Adds the current KTML state to the undo queue.

Method name	Action to perform
openColorPicker (o,t)	Opens the ***Color Picker*** either for the **Background Color** or **Foreground Color** buttons. *The parameters are:* • ***o*** – object = the caller HTML object. • ***t*** – type. The ***Color Picker*** will be positioned relatively to the object that will be opened. The **t** parameter can have either the ***f*** value (for foreground) or the ***b*** one (for background). *Example:* `<input type="button" Value="Open color picker" onclick="ktml_cont_cnt.openColorPicker(this, 'b')">`
getSelectedNode	Returns the HTML object/node containing the selection.
removeSelectedNode	Removes the HTML object/node containing the selection.

insertObject(newObject)	Inserts a newly created HTML element into the document. It has one parameter – the HTML element.
	
	Example:
	`var newElement = ktmlObject.edit.createElement("TT");`
	`newElement.innerText = "This is newly created TT element!";`
	`ktmlObject.insertObject(newElement);`

insertText(text)	Replaces the current selection with the desired text.
insertHTML(htmlText)	Replaces the current selection with the desired HTML text.
getHTMLContent	Returns the current HTML content.
getXHTMLContent	Returns the XHTML compliant content.
applyCSSStyle(styleName)	Applies the desired style on the current selection. It has one parameter – the style name.
applyHeadingStyle(headingName)	Applies the desired heading on the current selection. It has one parameter – the heading name.
openInsertTableDLG	Opens the insert table dialog pop-up window.
expandSelectionToParentNode	Selects the parent node of the current selected node.

These methods will help you customize the KTML object in order to suit your needs.

KTML Toolbar

The **KTML** toolbar is configurable and extensible, meaning you can create custom buttons in order to add features to the KTML object.

By using the API exported by KTML and the toolbar extensibility you can customize the KTML object behavior in order to suit your needs. To each button you can attach one of the already existing KTML object methods or you can define your own custom functions.

The Selection Mechanism

You - the KTML programmer - can access the current selection from the **KTML** editable area by using the `getSelectedNode` method of the KTML object.

This will be very helpful if you want to get an image or table's properties in order to use them further on.

In order to replace the current selection with:

- text – use the `insertText` method.
- HTML code – use the `insertHTML` method.
- an object – use the `insertObject` method.

Built-in Toolbar Buttons

The list of built-in toolbar buttons is defined by the `ktmlButtons` variable in the `toolbar.js` file. Thus, by using the `setToolbarButtons` function you will be able to display only the desired built-in buttons in the toolbar.

The KTML built-in toolbar buttons are: **Cut**, **Copy**, **Paste**, **Undo**, **Redo**, **InsertTable**, **BulletList**, **NumberedList**, **Indent**, **Outdent**, **HorizontalLine**, **AlignLeft**, **AlignCenter**, **AlignRight**, **AlignJustify**, **ToggleVisible**, **ToggleEditMode**, **CleanHTMLContent**, **Help**, **HeadingList**, **Style List**, **Font Type**, **FontSize**, **InsertLink**, **Bold**, **Italic**, **Underline**, **ForegroundColor**, **BackgroundColor**.

```
ktmlButtons = [
  new ToolbarButton("Cut", null, null, "cutcopypaste"),
  new ToolbarButton("Copy", null, null, "cutcopypaste"),
  new ToolbarButton("Paste", null, null, "cutcopypaste"),
  new ToolbarButton("Undo", null, null, "undoredo"),
  new ToolbarButton("Redo", null, null, "undoredo"),

  new ToolbarButton("InsertTable", null, null, null, null, "img", null, KT_JS_STRING,
"this.ktml.logic_InsertTable()"),
  new ToolbarButton("BulletList", null, null, null, DECMD_UNORDERLIST, "img", null, KT_CMD_ID,
"InsertUnorderedList"),
  new ToolbarButton("NumberedList", null, "numberlist", null, DECMD_ORDERLIST, "img", null, KT_CMD_ID,
"InsertOrderedList"),
  new ToolbarButton("Indent", null, null, "indentation"),
  new ToolbarButton("Outdent", null, null, "indentation"),
  new ToolbarButton("HorizontalLine", null, "hr", "horline", DECMD_INSERTHR, "img", null, KT_CMD_ID,
"InsertHorizontalRule"),
  new ToolbarButton("AlignLeft", null, null, "paralign", DECMD_JUSTIFYLEFT, "img", null, KT_CMD_ID, "JustifyLeft"),
  new ToolbarButton("AlignCenter", null, null, "paralign", DECMD_JUSTIFYCENTER, "img", null, KT_CMD_ID,
"JustifyCenter"),
  new ToolbarButton("AlignRight", null, null, "paralign", DECMD_JUSTIFYRIGHT, "img", null, KT_CMD_ID,
"JustifyRight"),
  new ToolbarButton("AlignJustify", null, null, "paralign", DECMD_JUSTIFYFULL, "img", null, KT_CMD_ID,
"JustifyFull"),
  new ToolbarButton("ToggleVisible", null, null, "toggleinvis", null, "img", null, KT_CMD_ID, "toggleInvisibles"),
  new ToolbarButton("ToggleEditMode", null, "html", "toggleedit", null, "img", null, KT_CMD_ID, "toggleEditMode"),
  new ToolbarButton("CleanHTMLContent", null, "cleanword", "cleanup", null, "img", null, KT_CMD_ID, "CleanHTML"),

  new ToolbarButton("Help", null, null, null, null, "img", null, KT_JS_STRING, "if ( pageLoded() &&
this.ktml.toolbar.getHelpMode()) {this.ktml.toolbar.setHelpMode(false, this);} else {
this.ktml.toolbar.setHelpMode(true, this.btn); }"),
  new ToolbarButton("BREAK"),

  new ToolbarButton("HeadingList", null, "property_heading", "html_tags", DECMD_SETBLOCKFMT,"select",{
'<P>' : 'Normal',
'<p>' : 'Paragraph',
'<h1>' : 'Heading 1',
'<h2>' : 'Heading 2',
'<h3>' : 'Heading 3',
```

Figure 7 The list of built-in toolbar buttons

Creating a Custom Toolbar Button

A button is an object created with the `ToolbarButton()` function.

The ToolbarButton Syntax

```
function ToolbarButton(bName, bAlt, bId, helpId, cid, buttonType, optionsObject, commandType, command)
```

This function parameters are:

- ***bName*** – the new button name.
- ***bAlt*** – the alternative text that will be displayed when placing the mouse over the button.
- ***bId*** – the button ID which will also be used to name the icon existing in `ktml/images/editor_images`.
- ***helpId*** – the help file name which should be found in `ktml/includes/ktedit/help/`.
- ***cid*** – edit command identifier, used by browser to query the state value for the button (enabled/disabled). For a list of supported command identifiers see `ktml/includes/ktedit/utils3.js`.
- ***buttonType*** – sets the button type. It may be one of the following:
  - ***img*** – displays an image.
  - ***select*** – displays a drop-down menu with options from the ***optionsObject***.
- ***optionsObject*** – the options from the toolbar button having the type ***select***. This object contains (value,label) pairs. See an example in the below image:

```
 new ToolbarButton("HeadingList", null, "property_heading", "html_tags", DECMD_SETBLOCKFMT,"select",{
'<P>' : 'Normal',
'<p>' : 'Paragraph',
'<h1>' : 'Heading 1',
'<h2>' : 'Heading 2',
'<h3>' : 'Heading 3',
'<h4>' : 'Heading 4',
'<h5>' : 'Heading 5',
'<h6>' : 'Heading 6',
'<address>' : 'Address',
'<pre>' : 'Formatted'  }
```

Figure 8 Creating a "select" toolbar button along with the options

- ***commandType*** – there are 3 types of commands:
  - ***KT_CMD_ID*** – specifies that the toolbar button should call one of the commands supported by the KTML object, the name of the command is specified by the command parameter. A list of supported KTML commands IDs is defined by the `Ktml_logic_doFormat(what)` function in `ktml/includes/ktedit/ktml.js`.

```
function Ktml_logic_doFormat(what) {
  var toundo = true;
  switch (what) {
    case 'Cut':
      try {
        this.edit.execCommand(DECMD_CUT,OLECMDEXECOPT_DODEFAULT, null);
      } catch(e) {
        if (confirm('Unprivileged scripts cannot invoke the Cut, Copy, and Paste commands!\nPress OK to open a technical note window with possible explanations to your problem. Press Cancel to return to the editor.')) {
          var wnd = window.open('http://mozilla.org/editor/midasdemo/securityprefs.html', 'securityprefs', '');
          wnd.focus();
        }
      }
      break;
```

*Figure 9 The **Cut** command ID defined by the **Ktml_logic_doFormat** function*

- ***KT_JS_STRING*** – specifies the JavaScript custom code to be executed by the toolbar button. To access the KTML object to which this button is attached to use `this.ktml`.

```
ktml_cont_cnt.addToolbarButton(new ToolbarButton("TextInsertion", "TextInsertion", "TextInsertion",
 null, null, "img", null, KT_JS_STRING, 'this.ktml.insertHTML(prompt("Enter the text!","Text"));'));
```

*Figure 10 Using **this.ktml** to access the KTML object to which a button is attached*

- ***KT_JS_FUNCTION*** – specifies the function to be executed.

Example:

```
ktml_cont_cnt.addToolbarButton( new ToolbarButton("st", "st", "st",
null, null, "img", null, KT_JS_FUNCTION, customFunction));

function customFunction()

{

var selectedNode  = ktml_cont_cnt.getSelectedNode();
selectedNode.style.textDecoration="line-through";

}
```

- ***command*** – specifies the command to be executed.

Example:

- If ***commandType*** is ***KT_CMD_ID*** then the ***command*** parameter may be `InsertOrderedList`.

```
ktml_cont_cnt.addToolbarButton(new ToolbarButton("AlignCenter", null, null, "paralign", DECMD_JUSTIFYCENTER, "img",
 null, KT_CMD_ID, "JustifyCenter"));
```

*Figure 11 Example of **command** parameter when **comandType** is **KT_CMD_ID***

- If ***commandType*** is ***KT_JS_STRING*** then the ***command*** parameter may be `this.ktml.logic_doFormat('InsertStyle',this.value)`.

```
ktml_cont_cnt.addToolbarButton(new ToolbarButton("TextInsertion", "TextInsertion", "TextInsertion",
 null, null, "img", null, KT_JS_STRING, 'this.ktml.insertHTML(prompt("Enter the text!","Text"));'));
```

*Figure 12 Example of **command** parameter when **comandType** is **KT_JS_STRING***

- If ***commandType*** is ***KT_JS_FUNCTION*** then the ***command*** parameter may be a custom function.

```
ktml_cont_cnt.addToolbarButton( new ToolbarButton("st", "st", "st", null, null, "img", null, KT_JS_FUNCTION,
customFunction));
```

*Figure 13 Example of **command** parameter when **comandType** is **KT_JS_FUNCTION***

You will find below an example of how to create a custom toolbar button:

```
var btnInsertCustomHTML=new ToolbarButton("TextInsertion", "TextInsertion",
"TextInsertion",      null,      null,      "img",      null,      KT_JS_STRING,
'this.ktml.insertHTML(prompt("Enter the text!","Text"));');
```

After having created the button, you should add it in order to be displayed in the KTML toolbar.

Adding a Custom Toolbar Button

In order to add a custom toolbar button you should use:

```
ktml_cont_cnt.addToolbarButton(customToolbarButton);
```

There are two ways of adding a button:

1. Adding the button after it was created.
   - First, you create the button:

```
var btnInsertCustomHTML=new ToolbarButton
("expandSelectionToParentNode", "expandSelectionToParentNode",
"expandselectiontoparentnode", null, null, "img", null,
KT_JS_STRING, 'this.ktml.insertHTML(prompt("Enter the
text!","Text"));');
```

   - Then, you add it:

```
ktml_cont_cnt.addToolbarButton(btnInsertCustomHTML);
```

2. Create the button and adding it in the same time.

```
ktml_cont_cnt.addToolbarButton(new ToolbarButton
("expandSelectionToParentNode", "expandSelectionToParentNode",
"expandselectiontoparentnode", null, null, "img", null,
KT_JS_STRING, 'this.ktml.insertHTML(prompt("Enter the
text!","Text"));'));
```

After having inserted a code bloc similar to one of the ones presented above, the newly created button will be displayed in the KTML toolbar.

KTML Plug-in Mechanism

KTML was designed to be independent of the web server. However, some specific functionalities such as image gallery and spell-checker depend on server side code and data.

KTML is very extesible allowing you to attach such plug-ins.

In **KTML** all plug-ins are located in the *ktml/includes/ktedit/plugins/* folder.

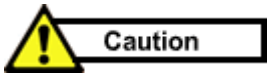
Plug-in Architecture

Each plug-in must include the following files:

1. *plugin.<<server_side_extension>>* (.php, .asp, .jsp, .pl) which will register plug-in properties into the KTML security session variable and generate all other HTML and JavaScript necessary code to extend the KTML object features.

2. *scripts.js* – which contains the definition of the JavaScript class associated to the plug-in, and some plug-in properties and methods. You can extend the KTML class by adding your own methods.

3. *index.<<server_side_extension>>* (.php, .asp .jsp, .pl) – or any name you may want to call it, where you will include the KTML object instance.



When wanting to use plug-ins, the web page which should include the KTML object should contain server-side code, therefore it should be dynamic (having an extension such as: ***.php***, ***.asp***, ***.jsp***, ***.pl*** or others)

Creating a Plug-in Sample

In order to make you understand what is the mechanism of a plug-in creation we will provide a **PHP** plug-in example. The plug-in whose creation process will be presented further on, is supposed to help the KTML user to insert in the editing area the name of a user selected from a database located on the server. The user will click on a toolbar button which will open a window which should allow the selection of a record (a user name). First the editor (the person who edits content) is supposed to type the first letters of the user name in a prompt window, then to select a user name from the displayed list. The selected user name will be included in the **KTML** editable area.

To create the plug-in, follow the steps presented below:

1. Create a new folder named **PluginSample** in *ktml/includes/ktedit/plugins/*.

2. In *ktml/includes/ktedit/plugins/PluginSample/* create a *plugin.php* file which will contain the plug-in initialization function:

```
<?php

function PluginSample($name)

{echo "<script>includeJS
(DIR_DEPTH+'includes/ktedit/pluggins/PluginSample/scripts.js');\r\n";

echo "ktml_$name.registerPluggin('PluginSample', new PluginSample
(ktml_$name));\r\n";

echo "ktml_$name.addToolbarButton(new ToolbarButton(\"InsertUser\", \"Insert
User\",     null,     null,     null,     \"img\",     null,     KT_JS_STRING,
\"this.ktml.openSelectUser()\"));";
```

```
echo "</script>";}
?>
```

3. In *ktml/includes/ktedit/plugins/PluginSample/* create *UserDialog.php* which will allow the selection of a user name from the database on the server.
   - First include the database connection file and the *rsUser* recordset used to select the users from the database:

```
<?php require_once('../../../../../Connections/dbconnection.php'); ?>
<?php
$colname_rsUser = "";
if (isset($HTTP_GET_VARS['pluginParam']))
{
  $colname_rsUser = (get_magic_quotes_gpc()) ? $HTTP_GET_VARS['pluginParam']
: addslashes($HTTP_GET_VARS['pluginParam']);
}
mysql_select_db($database_dbconnection, $dbconnection);
$query_rsUser = sprintf("SELECT id_usr, firstname_usr, concat
(firstname_usr,' ',lastname_usr) as full_name FROM users_usr WHERE
firstname_usr like '%s%%'", $colname_rsUser);
$rsUser = mysql_query($query_rsUser, $dbconnection) or die(mysql_error());
$row_rsUser = mysql_fetch_assoc($rsUser);
$totalRows_rsUser = mysql_num_rows($rsUser);
?>
```

   - Then, define how the interface should look like:

```
<html><head><title>Select user</title>
<script>
//to execute when the user clicks on the "Select User" button
function selectUser()
{
window.opener.ktmls[0].insertUserName(document.getElementById("user_list").
value);
window.close();
}
</script>
</head>
<body style="background-color:buttonface">
<?php
//if records found display the users' names drop-down menu and the "Select
User" button
if($totalRows_rsUser>0){?>
<select name="user_list" id="user_list" size="8">
<?php do
{?>
```

```
    <option value="<?php echo htmlspecialchars($row_rsUser['full_name']);?
>"><?php echo $row_rsUser['full_name']?></option>
<?php
} while ($row_rsUser = mysql_fetch_assoc($rsUser));
  $rows = mysql_num_rows($rsUser);
  if($rows > 0) {
      mysql_data_seek($rsUser, 0);
             $row_rsUser = mysql_fetch_assoc($rsUser);
}
?>
</select>
<input type="button" value="Select" onclick="selectUser();">
<?php
}else{
//if there are no user names matching the criterion, just display a message:
?>
No user has been found with name starting with '<?php echo
$colname_rsUser;?>'
<?php
}
?>
//displaying a "Close window" button
<input type="button" value="Close window" onclick="window.close();">
</body></html><?php mysql_free_result($rsUser);?>
```

4. In *ktml/includes/ktedit/plugins/PluginSample/* create *scripts.js* which will define the plug-in JavaScript class and extend the KTML class with two more methods:
   - The opening method of the dialog window which can be attached to a custom toolbar button.
   - The insertion method in the KTML edited area of the selected user name.

```
/**
*the PluginSample class
*/
function PluginSample(owner)
{
this.ktml = owner;
return this;
}
/**
*Opens the select user dialog
*/
function Ktml_openSelectUser()
```

```
{
    var pluginParam=prompt("Enter the starting letter:", "");
            pluginParam = pluginParam.substring(0,1);
                    util_openwnd("windowName",
DIR_DEPTH+"includes/ktedit/pluggins/PluginSample/UserDialog.php?pluginParam=
"+pluginParam, 300, 200);
}
/**
*Insert the name of the selected user into the document, replacing the
selection
*/
function Ktml_InsertUserName(userName)
{
this.insertText(userName);
this.cw.focus();
}
//open the plug-in dialog window
Ktml.prototype.openSelectUser = Ktml_openSelectUser;
//insert the selected user into the edited document
Ktml.prototype.insertUserName = Ktml_InsertUserName;
```

5. Call the `PluginSample` function including the plug-in from `index.php` after having set the KTML properties and after having added the custom buttons. So, the following code lines should be inserted after the `<script>` tag that contains the KTML object instance definition, the KTML object properties, the built-in and custom toolbar buttons definitions.

```
<?php
 include('ktml/includes/ktedit/pluggins/PluginSample/plugin.php');
                    PluginSample("cont_cnt");
?>
```

In the browser you will notice the display of another toolbar button (without an icon) named **Insert User** (as you will be able to see when moving the mouse over the button). After clicking on this button the following prompt window will be displayed:

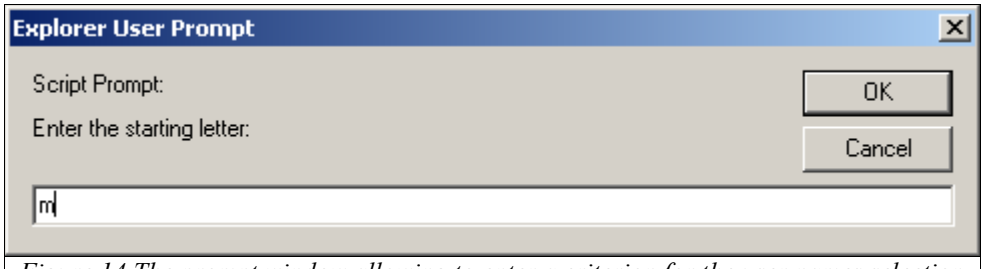


Figure 14 The prompt window allowing to enter a criterion for the user names selection

After having entered a letter or a string of characters and clicked on **OK**, a pop-up window similar to the one below will be displayed if there are record in the database matching your criterion:

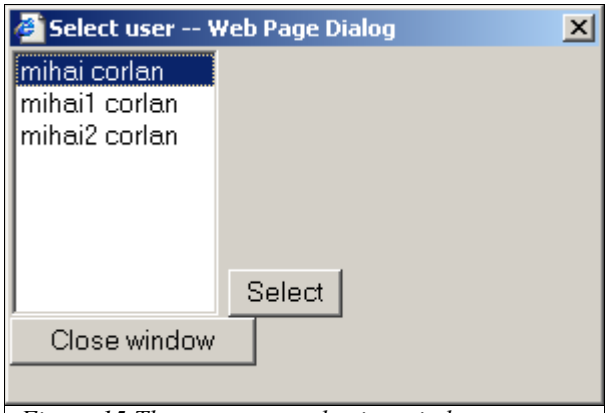


Figure 15 The user name selection window

Select a user name and click on the **Select** button. The user name will be inserted in the KTML editable area:

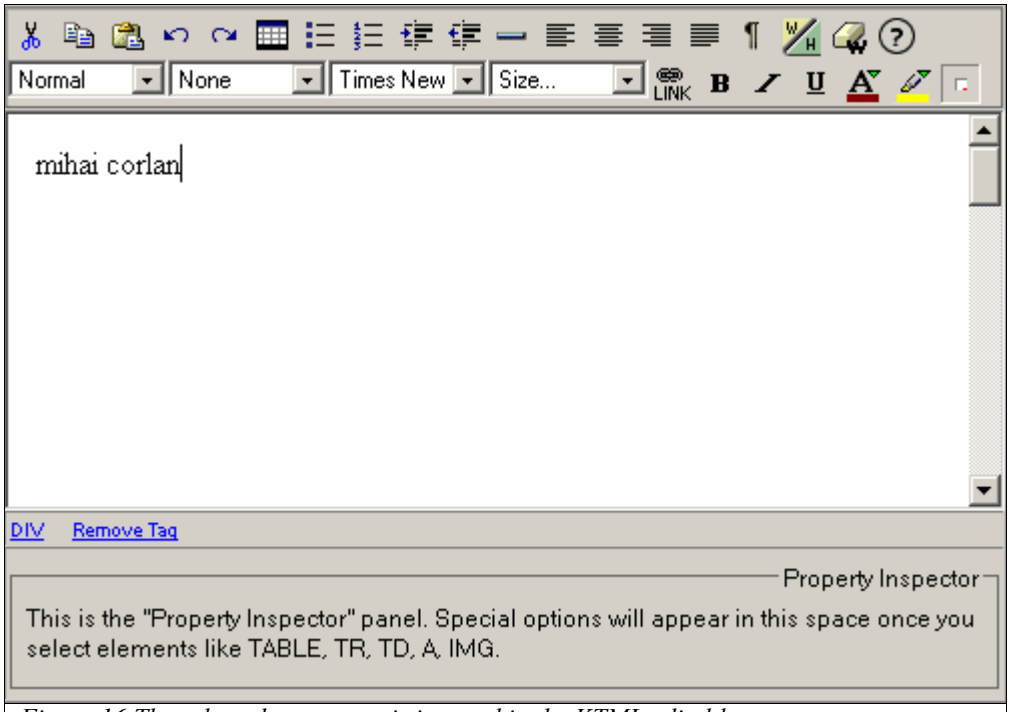


Figure 16 The selected user name is inserted in the KTML editable area

Based on this example, you should be able to create your own plug-ins and to attach them to your customized KTML object.

Conclusions

This document is of great help for when you will try to customize and extend the **KTML** - HTML editor.

After having read it you will be able to include a KTML object instance in your web page, to set the KTML properties, and to select the built-in buttons that will be displayed in the toolbar. You will also find out what are the API methods that can be attached to a button in order to extend the KTML toolbar.

By following the steps described in the document, you will also be able to create your own server-side plug-ins and to attach them to KTML.

Further Reading

For comprehensive information on all features of **KTML**, we recommend reading **KTML User Manual**.

In order to train your editors on how to use the **KTML** please refer to **Publishing Content with KTML Tutorial**.

Copyrights and Trademarks

Copyright - 2000-2004 by InterAKT Online, SRL.

All Rights Reserved. This tutorial is subject to copyright protection.

KTML, MX Kart, MX Shop, MX Includes, MX Looper, PHAkt, ImpAKT, NeXTensio, MX Query Builder, Transaction Engine are trademarks of InterAKT.

All other trademarks are acknowledged as the property of their respective owners.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of this document or of the associated product may be reproduced in any form by any means without prior written authorization of InterAKT Online, except when presenting only a summary of the tutorial and then linking to the InterAKT website.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIEDWARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Send comments and suggestions to products@interaktonline.com.

Interakt

professional web tools

InterAKT Online
web: http://www.interakt.ro/
e-mail: contact@interakt.ro

tel: +4021 312 5 312